Discrete Applied Mathematics 159 (2011) 1418–1424

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Minimum cost open chain reconfiguration

Ali Nourollah [a,*], Mohammadreza Razzazi [b]

[a] Department of Electrical and Computer Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran

[b] Software Systems R&D Lab., Department of Computer Engineering & IT, Amirkabir University of Technology, #424 Hafez Avenue, P. O. Box 15875-4413, Tehran, Iran

ARTICLE INFO

Article history:
Received 20 February 2008
Received in revised form 26 April 2011
Accepted 16 May 2011
Available online 2 July 2011

Keywords:
Open chain reconfiguration
Robot reconfiguration
Dynamic programming
Ruler folding problem

ABSTRACT

An open chain or n-link is a sequence of n links with fixed lengths that are joined together at their endpoints and can turn about their endpoints, which act as joints. Positions of the joints of a chain define a configuration of the chain in the space. In one-dimensional space, we define a binary configuration as a sequence of direction of links. Open chain reconfiguration is a sequence of predefined transformation operations which can be used to convert a given binary configuration to another given binary configuration. Each transformation operation is assigned a cost. For two given binary configurations, there may be many reconfigurations whose costs are different. We formalize the problem, and we propose a dynamic programming approach to find a reconfiguration whose cost is minimum for the conversion of two given binary configurations of an open chain in the one-dimensional space. Our algorithm takes $O(n^2)$ time using $O(n)$ space.

1. Introduction

Serpentine robots, also sometimes called *snake robots*, are slender, multi-segmented vehicles designed to provide greater mobility than conventional wheeled or tracked robots. Snake robots typically comprise of three or more rigid segments that are connected by 2- or 3-degrees-of-freedom joints. The segments typically have powered wheels, tracks, or legs to propel the vehicle forward, while the joints may be powered or unpowered [5]. In this paper, a problem related to the movements of a multilink robot is considered. Snake-like robots are very useful and flexible in many applications such as rescue in earthquake. Shan designed a snake-like robot which, with some modifications, is appropriate for our algorithm. He presents the design and motion planning for a mechanical snake robot that was built at the University of Michigan. The structure of the robot enables it to move without wheels. It is constructed of a series of articulated links, each one with a motor and a linear solenoid. Although each link has only one motor, this structure allows the body configuration to be easily controlled, thereby enabling the robot to move in very cluttered environments. The motion-planning system provides the robot with a basic motion pattern that can be easily modified for different tasks and environments. The mechanical snake does not avoid obstacles on its way, but rather "accommodates" them by continuing its motion toward the target while in contact with the obstacles. Each link has a different number of degrees-of-freedom in each motion stage, providing the robot with great adaptability even during contact with obstacles in a cluttered environment. His robot is shown in Fig. 1 [16,17,4].

To describe our algorithm, we consider an open chain as an abstract representation of a snake-like robot. An open chain is constructed from n links, which are straight line segments of arbitrary real lengths. These links are joined together end-to-end by freely rotating joints so that the links are allowed to cross each other. The configuration of an open chain is specified by the positions of all the joints. In many applications such as vehicle manufacturing, one end point of the open chain is hinged.

* Corresponding author. Fax: +98 21 23693221.
E-mail addresses: nourollah@aut.ac.ir (A. Nourollah), razzazi@aut.ac.ir (M. Razzazi).

0166-218X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2011.05.005

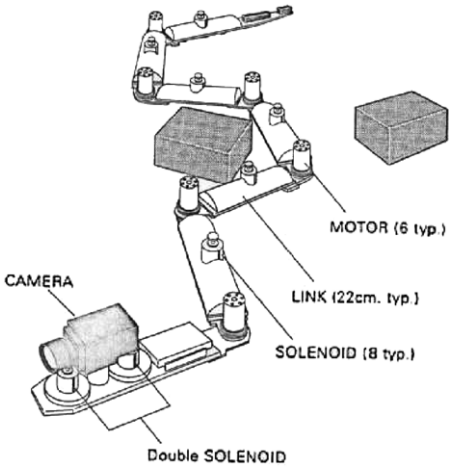

Fig. 1. Shan's snake robot.

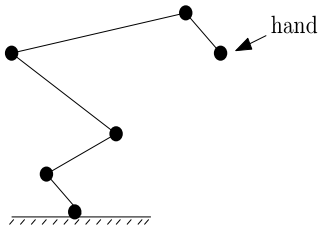

Fig. 2. A robot arm.

In this case, the open chain is referred to as a *robot arm*. In many applications that a robot needs to fold for manipulating an object, minimum folding time implies increasing system's throughput (rate of production over a stated period of time). Fig. 2 shows a robot arm in which one end point is fixed using a pin. The other free end point is called *hand*.

Kantabutra introduced a linear time algorithm to determine whether a given configuration of an n-link short-linked open chain confined in a square can be transformed to another given configuration[8]. His paper represents a step toward understanding the computational complexity of motion-planning problems. One of the first groups of researchers to study the computational complexity of motion planning was Hopcroft et al., who had a considerable amount of success on the study of open chains of the type described above[6,7]. In [6], they showed that reachability decision problems are NP-hard if the open chain is constrained by arbitrary polygonal walls. However, when the boundary is a circle, then they were able to give an algorithm that determines the reachability of any open chain configuration from any other configuration in $O(n)$ time. They also gave an algorithm of $O(n^3)$-time complexity that moves the open chain to any specific reachable configuration by $O(n^3)$ link moves. In [10], Kantabutra and Kosaraju presented a more efficient solution, involving only $O(n)$ time and $O(n)$ link moves. In [7], Hopcroft et al. developed a polynomial-time algorithm for computing the regions reachable by the joints of an open chain in a circle. This polynomial is at least quadratic in the input size. Kantabutra and Kosaraju presented in [10] a linear time solution to this problem.

A known problem which is related to chain reconfiguration is the "*Ruler Folding Problem*". The aim of this problem is to find the minimum length of folded chain in which each joint is to be completely open, or completely folded. Although several algorithms exist which solves the "Ruler Folding Problem" and generate a configuration of an open chain, none of them considers the minimization of the cost of reconfiguration.

"*Ruler Folding Problem*" was introduced by Hopcroft et al. for the first time, and has been shown to be NP-complete by a reduction from PARTITION problem [6]. They developed an $O(nL^2)$ pseudo-polynomial-time algorithm for optimal folding of an n-link open chain in one-dimensional space, where L is the length of the longest link [6,18]. Hopcroft et al. proposed a linear time approximation algorithm for the "*Ruler Folding Problem*" with the upper bound of $2L$ for the length of a folded chain, where L is the length of the longest link of the chain. They showed that this upper bound is tight using an example [6].

Reachability is a class of robotic problems. In many applications snake-like robots should be folded to reach a given point; and in many cases, the only way to reach a point is folding up very compactly. Fig. 3 shows an abstracted snake-like robot which needs to be folded to reach a point p. This figure is an example that is used to prove that the reachability problem is NP-Complete. In this figure, since the width of the tunnel is very narrow, not every link of the robot can rotate in the tunnel and the robot cannot reach a point p via rotation solution. The only solution for this sample is folding the robot and moving it right through the tunnel, then moving upward, and moving to the left. For this solution, value x should be greater than

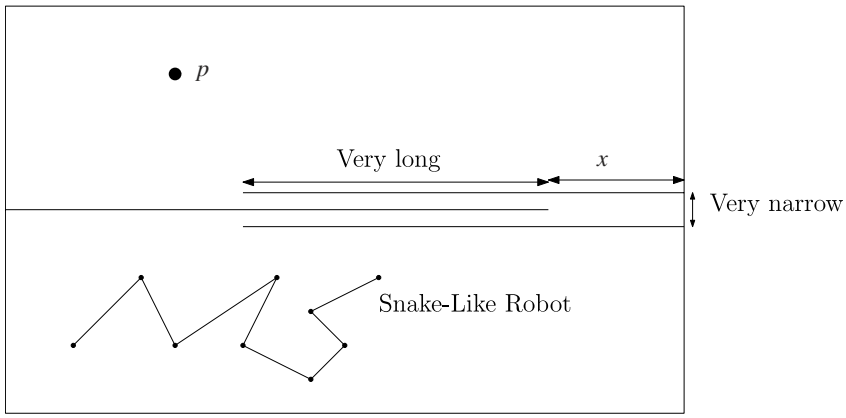

Fig. 3. An snake-like robot.

folded length of the robot. Thus, if the length of the folded robot is less than the value of x, the robot can reach the point p. This example also indicates that reachability problem is reduced to a ruler folding problem and is NP-Complete.

Recently, Calinescu and Dumitrescu improved the previous results and provided a fully polynomial-time ϵ-approximation scheme for the ruler folding problem[3]. Total running time of their algorithm is $O(n^4(1/\epsilon)^3 \log L)$ and requires $O(n^4(1/\epsilon)^3 \log L)$ additional space. Generalized open chains such as trees and graphs are called linkages. Works on linkages other than folding are given in [9,1,2,11,19,15].

In [12], Nourollah and Razzazi introduced the ruler folding problem in d-dimensional space. They proposed a dynamic programming approach, using $O(nm_1)$ time and space, to fold a given chain into a minimum length, assuming that the links have integer lengths. They also showed that by generalizing the algorithm, it can be used to solve the *orthogonal ruler folding problem* in d-dimensional space, such that it requires $O(2^d ndm_1^d)$ time using $O(2^d ndm_1^d)$ space. Furthermore in [13], they introduced a new linear time approximation algorithm for ruler folding problem which it folds the given chain in $\max\{2m_1 - \frac{m_1-m_2}{2^{k-2}}, \sum_{i=0}^{k-1} \frac{m_1}{2^i}\}$, where m_1 and m_2 are the lengths of the two distinct maximum length links in the chain respectively, and k is the number of links whose lengths are m_1. In [14], Nourollah and Razzazi introduced the design of a snake-like robot which could be folded in a given interval.

To our knowledge, no paper related to folding or reconfiguration of an open chain has considered cost of the links movement. However, in the real world, the cost of the movement of different links is not the same. For example in real robots, weight of the links affects the cost, and weight of the links is related to the length of the links. In the graphical applications such as linkage's animations, the number of moving links affect the cost. In this paper, for the first time we present a dynamic programming algorithm, which takes $O(n^2)$ time using $O(n)$ space, to find the minimum cost to reach a given configuration of an n-link open chain in one dimension from another given configuration of the open chain. Preliminaries are stated in Section 2. In Section 3, the folding algorithm is given. Realizing our algorithms is presented in Section 4, and the conclusion is stated in Section 5.

2. Preliminaries

A *linkage* is a planar straight line graph $G = (V, E)$ and a mapping $l : E \longmapsto R^+$ of edges to positive real lengths. Each vertex of a linkage is called a *joint* or an *articulation point*, each straight line edge e of a linkage which has a specified fixed length $l(e)$ is called a *bar* or a *link*. A linkage whose underlying graph is a single path is called *polygonal arc*, *open chain* or a *ruler*; a linkage whose underlying graph is a single cycle is called *polygonal cycle*, *closed chain* or a *polygon*; and a linkage whose underlying graph is a single tree is called *polygonal tree* or *tree linkage*. An n-link polygonal arc denoted by $\Gamma_{1,n}$ is a sequence of n *links* of arbitrary finite lengths moving in Euclidean plane. These links are joined together end-to-end by freely rotating *joints*. The joints are denoted by A_k, for $k = 0, \ldots, n$. The link between A_{k-1} and A_k, $k = 1, \ldots, n$, is called l_k. The length of a link l is shown by $|l|$. Assuming that all joints are located on x-axis, the position of joint A_k, $k = 0, \ldots, n$, on the x-axis is denoted by J_k.

Let i and n be two positive integers such that $i \leq n$, two sets $\{i, i+1, \ldots, n\}$ and $\{i-1, i, i+1, \ldots, n\}$ is denoted by $\mathbb{I}_{i,n}$ and $\mathbb{I}^+_{i,n}$ respectively. For an open chain $\Gamma_{i,n} = \langle l_i, l_{i+1}, \ldots, l_n\rangle$, an ordered tuple $C_{i,n} = \langle J_{i-1}, J_i, \ldots, J_n\rangle$ is referred to as a *configuration* in which

$$\forall k \in \mathbb{I}_{i,n} : J_k \in \mathbb{R} \quad \text{and} \quad |J_k - J_{k-1}| = |l_k| \quad \text{such that } |l_k| > 0.$$

A configuration realizes an open chain in the one-dimensional space. Space of all configurations of $C_{i,n}$ is denoted by $\mathcal{C}_{i,n}$. For each configuration $C_{i,n} = \langle J_{i-1}, J_i, \ldots, J_n\rangle$, a *binary configuration* $B_{i,n} = \langle f_i, \ldots, f_n\rangle$ is defined as follows.

$$\text{for } k \in \mathbb{I}_{i,n} : f_k = \begin{cases} +1 & J_k > J_{k-1}, \\ -1 & J_k < J_{k-1}. \end{cases} \tag{1}$$

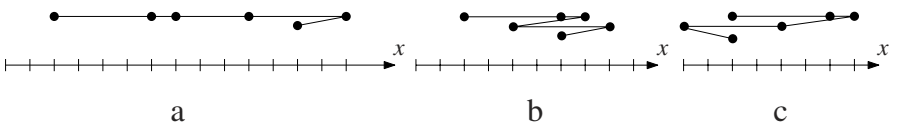

Fig. 4. (a) An open chain Γ whose binary configuration is $B_{1,5} = (+1, +1, +1, +1, -1)$. (b) Transformed open chain Γ after applying function $Turn(B_{1,5}, 3) = (+1, +1, -1, +1, -1)$. (c) Transformed open chain Γ after applying function $Rot(B_{1,5}, 3) = (+1, +1, -1, -1, +1)$.

Note that the binary configuration realizes the folding of a given open chain and the positions of joints are not considered. Space of all binary configuration of $B_{i,n}$ is denoted by $\mathcal{B}_{i,n}$.

We define *transformation function* as a function which maps a binary configuration to another binary configuration, and we introduce two transformation functions *Turn* and *Rot* transforming a given binary configuration to another binary configuration.

Note that, to change one given binary configuration to another given binary configuration, our algorithm starts from one end point of the linkage and applies transformation functions on sequent the other end of the linkage. For doing this, independent of an end point of a robot being fixed or not, no Shift operation is needed.

Let $\Gamma_{i,n} = (l_i, l_{i+1}, \ldots, l_n)$ be an $(n - i + 1)$-link open chain whose configuration and binary configuration are given by $C_{i,n} = \langle J_{i-1}, J_i, \ldots, J_n \rangle$ and $B_{i,n} =< f_i, \ldots, f_n >$ respectively. For the given binary configuration $B_{i,n}$ and a link index $k \in \mathbb{I}_{i,n}$, function $Turn : \mathcal{B}_{i,n} \times \mathbb{I}_{i,n} \longmapsto \mathcal{B}_{i,n}$ turns the kth link of the open chain, $k \in \mathbb{I}_{i,n}$, to the opposite direction and it is defined as follows.

$$Turn(\langle f_i, \ldots, f_n \rangle, k) = \langle f_i, \ldots, f_{k-1}, -f_k, \ldots, f_n \rangle.$$

Function $Rot : \mathcal{B}_{1,n} \times \mathbb{I}_{i,n} \longmapsto \mathcal{B}_{1,n}$ rotates the subchain $\langle l_k, \ldots, l_n \rangle$ from its first point, J_{k-1}, and is defined as follows.

$$Rot(\langle f_i, \ldots, f_n \rangle, k) = \langle f_i, \ldots, f_{k-1}, -f_k, \ldots, -f_n \rangle.$$

Fig. 4 shows an example of applying turn and rotate functions on an open chain.

Let B be a given binary configuration and let g and f be two transformation functions. Applying g and f respectively is combination of functions f and g and it is denoted by $f \circ g$. Let B be a binary configuration and let f, g, and h be three transformation functions. It is easy to see that $f \circ g = g \circ f$ and $(f \circ g) \circ h = f \circ (g \circ h)$.

Transformation of binary configuration B to $\hat{B}$ by one transformation function is denoted by $B \vdash \hat{B}$, and transformation of binary configuration B to $\hat{B}$ by zero or more transformation functions is denoted by $B \vdash^* \hat{B}$. Transformation of binary configuration B to $\hat{B}$ by a transformation function g is denoted by $B \vdash^g \hat{B}$. It is easy to see that for a given binary configuration $B_{i,n} \in \mathcal{B}_{1,n}$, $i \in \mathbb{I}_{1,n}$, and a link index $k \in \mathbb{I}_{i,n}$, $B_{i,n} \vdash^{Turn(k)} B'_{i,n} \vdash^{Turn(k)} B_{i,n}$, $\forall k \in \mathbb{I}_{i,n}$. Let B and B' be two binary configurations, and g be a transformation function for which $B \vdash^g B'$, *cost of transformation* of g is denoted by $Cost(g)$. Note that $Cost(g)$ has various values in mechanical or graphical applications.

Let $B^I_{i,n}$ and $B^F_{i,n}$ be an initial and final binary configuration respectively, and let $G = \langle g_1, \ldots, g_k \rangle$ be a sequence of k operations which it transforms the configuration $B^I_{i,n}$ to $B^F_{i,n}$ such that

$$\exists B^1_{i,n}, \ldots, B^{k-1}_{i,n} : B^I_{i,n} \vdash^{g_1} B^1_{i,n} \vdash^{g_2} B^2_{i,n} \vdash^{g_3} \ldots \vdash^{g_{k-1}} B^{k-1}_{i,n} \vdash^{g_k} B^F_{i,n}.$$

In this case, transforming $B^I_{i,n}$ to $B^F_{i,n}$ using G is denoted by $B^I_{i,n} \vdash^G B^F_{i,n}$ and is referred to as *reconfiguration* of $B^I_{i,n}$ to $B^F_{i,n}$. *Transformation cost of* G is denoted by $Cost(G)$ and is defined as

$$Cost(G) = \sum_{i=1}^{k} Cost(g_i).$$

For a given binary configuration $B_{i,n} = \langle f_i, \ldots, f_n \rangle$, $Rot(B_{i,n}, i)$ is denoted by $\bar{B}_{i,n}$. In the next section, we introduce an algorithm to find minimum cost of reconfiguration for two given binary configurations of an open chain.

3. Our algorithm

Let $\Gamma_{1,n}$ be an n-link open chain whose two binary configurations $B_{1,n} = \langle f_1, \ldots, f_n \rangle$ and $\hat{B}_{1,n} = \langle \hat{f}_1, \ldots, \hat{f}_n \rangle$ are given. The problem is to find the minimum cost for reconfiguration of $B_{1,n}$ to $\hat{B}_{1,n}$. We design a dynamic programming approach to solve this problem.

Let $i \in \mathbb{I}_{1,n}$ and $k \in \mathbb{I}_{i,n}$, we define $MinCost(B_{i,n} \vdash^* \hat{B}_{i,n}, k)$ as the least cost for reconfiguration of $B_{k,n}$ to $\hat{B}_{k,n}$. It is easy to see that

$$\forall i \in \mathbb{I}_{1,n} : \quad MinCost(B_{i,n} \vdash^* \hat{B}_{i,n}, n + 1) = 0$$

and

$$\forall i \in \mathbb{I}_{1,n} : \quad MinCost(\bar{B}_{i,n} \vdash^* \hat{B}_{i,n}, n + 1) = 0.$$

We should find $MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, 1)$. We have

$$\forall k \in \mathbb{I}_{1,n},\ \forall j \in \mathbb{I}_{k,n}: \quad MinCost(B_{k,n} \vdash^* \hat{B}_{k,n}, j) = MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, j). \tag{2}$$

To solve the problem, we shall compute it by finding $MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k)$, $\forall k \in \mathbb{I}_{1,n}$.

Lemma 1. *Let $\Gamma_{1,n}$ be an n-link open chain whose two binary configurations $B_{1,n}$ and $\hat{B}_{1,n}$ are given, and j be an integer for which $j \in \mathbb{I}_{1,n}$. We have*

$$\forall k \in \mathbb{I}_{1,j}: \quad MinCost(Rot(B_{1,n}, k) \vdash^* \hat{B}_{1,n}, j) = MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, j).$$

Proof. Using Eq. (2), we have

$$\begin{aligned} \forall k \in \mathbb{I}_{1,j}:\ MinCost(Rot(B_{1,n}, k) \vdash^* \hat{B}_{1,n}, j) &= MinCost(Rot(B_{k,n}, k) \vdash^* \hat{B}_{k,n}, j) \\ &= MinCost(\bar{B}_{k,n} \vdash^* \hat{B}_{k,n}, j) \end{aligned}$$

$/* \text{Using Eq. (2)} */ = MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, j)$. □

The recursive definition of $MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k)$ is as follows.

$$MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k) = \begin{cases} MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k+1) & f_k = \hat{f}_k, \\ min\{X(k), Y(k)\} & f_k \neq \hat{f}_k, \end{cases} \tag{3}$$

where $B_{1,n} = \langle f_1, \ldots, f_n \rangle$ and $\hat{B}_{1,n} = \langle \hat{f}_1, \ldots, \hat{f}_n \rangle$,

$$X(k) = Cost(Turn(k)) + MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k+1),$$

and

$$Y(k) = min\{Cost(Rot(i)) + MinCost(Rot(B_{1,n}, i) \vdash^* \hat{B}_{1,n}, k+1);\ \text{for } i = 1, \ldots, k\}.$$

Using Lemma 1, we can rewrite $Y(k)$ as follows.

$$Y(k) = min\{Cost(Rot(i)) + MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, k+1);\ \text{for } i = 1, \ldots, k\}. \tag{4}$$

Furthermore,

$$MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, k) = \begin{cases} MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, k+1) & \bar{f}_k = \hat{f}_k, \\ min\{\bar{X}(k), \bar{Y}(k)\} & \bar{f}_k \neq \hat{f}_k, \end{cases} \tag{5}$$

where $\bar{B}_{1,n} = \langle \bar{f}_1, \ldots, \bar{f}_n \rangle = \langle -f_1, \ldots, -f_n \rangle$,

$$\bar{X}(k) = Cost(Turn(k)) + MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, k+1),$$

and

$$\bar{Y}(k) = min\{Cost(Rot(i)) + MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k+1);\ \text{for } i = 1, \ldots, k\}. \tag{6}$$

Pseudocode of the algorithm is shown in Fig. 5. In the algorithm,

$$MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k)$$

and

$$MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, k),$$

for $k = 1, 2, \ldots, n+1$, can be implemented by two arrays of size $n+1$, so the algorithm requires $O(n)$ space, and since computation of $Y(k)$ and $\bar{Y}(k)$ at most takes $O(n)$ time, so the total time of the algorithm is $O(n^2)$. To answer the optimality of the algorithm with respect to time or space, needs further investigation. Concerning the optimal solution, it can be obtained by using an extra array and utilizing the information provided by the algorithm. Note that for the optimal substructure, it is easy to see that an optimal solution to a problem contains within it an optimal solution to the subproblems.

Algorithm $MinCostReconfiguration(B, \hat{B})$

// It computes minimum cost reconfiguration of B to $\hat{B}$.

Let $MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, n+1) = 0$

Let $MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, n+1) = 0$

for $k \leftarrow n$ **downto** 1 **do**

Compute $MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, k)$ and

$MinCost(\bar{B}_{1,n} \vdash^* \hat{B}_{1,n}, k)$ according to equations (3), (4), (5), and (6)

return $MinCost(B_{1,n} \vdash^* \hat{B}_{1,n}, 1)$

End of Algorithm

Fig. 5. Minimum cost reconfiguration algorithm.

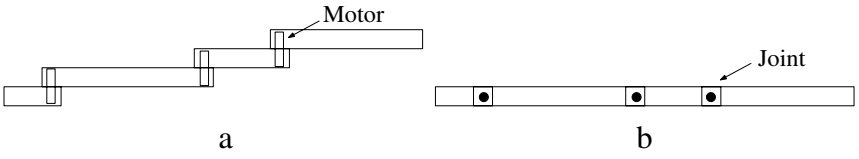

Fig. 6. Our proposed joints for which full rotation (0° . . . 360°) is possible. (a) Side view. (b) Up view.

4. Realizing the folding algorithm

In order for Shan's robot to be more practical in terms of accessibility and path planning two changes in the design needs to be done.

(a) Size of link. Depending on the application (e.g. fact finding for different situations which an earthquake may create) links with selected sizes should be used to constitute a snake robot. In other words, we should be able to replace any link with some other appropriately sized link. This can be done using any of the following approaches.
   (1) Each link consists of two links one sliding into another. Therefore, the length of each link can be modified dynamically.
   (2) There are a set of constructed fixed and different lengths links, and each robot can be constructed dynamically from them.

   This approach introduces *reconfigurable snake robots*. A reconfigurable robot can be used in many applications with high flexibility. If the lengths of the links can be modified dynamically, an efficient algorithm to fold the robot into the minimum length is useful to apply in practice.

(b) Joint. To be able to better move a robot between obstacles, as was mentioned in the Section 1, joints need to be designed so that the full folding of the links are possible. One solution which is a modification to joints of Shan's robot is given in the Fig. 6. In this approach links connected to a joint can rotate the full range of 0° . . . 360°.

A real robot challenges many problems, theoretically and practically. In many cases, motion-planning algorithms cannot solve the problem. In these cases, an efficient algorithm to fold the robot is applicable.

5. Conclusions

To our knowledge, no paper studying the problem of how to reach a given binary configuration of an open chain from any other binary configuration of the open chain has considered cost of the links movement. In this paper we introduced a dynamic programming algorithm which requires $O(n^2)$ time using $O(n)$ space to determine the minimum cost reconfiguration of an open chain. This algorithm can be used in robot motion planning and graphical applications such as linkage's animations. Furthermore, we introduced a practical idea to construct reconfigurable and resizable snake robots which can be folded. Time versus space trade-off of the algorithm needs further investigation.

References

[1] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, S. Robbins, I. Streinu, G. Toussaint, S. Whitesides, A note on reconfigurating tree linkages: trees can lock, Discrete Applied Mathematics (2001).
[2] T. Biedl, A. Lubiw, J. Sun, When can a net fold to a polyhedron? in: Eleventh Canadian Conference on Computational Geometry, U. British Columbia, 1999.
[3] G. Calinescu, A. Dumitrescu, The carpenter's ruler folding problem, in: Jacob Goodman, János Pach, Emo Welzl (Eds.), Combinatorial and Computational Geometry, Mathematical Sciences Research Institute Publications, Cambridge University Press, 2005, pp. 155–166.
[4] K.J. Dowling, Limbless Locomotion: Learning to Crawl with a Snake Robot, Ph.D. Thesis, The Robotics Institute Carnegie Mellon University, September, December, 1997.
[5] G. Granosik, J. Borenstein, Integrated joint actuator for serpentine robots, IEEE/ASME Transactions on Mechatronics 10 (5) (2005) 473–481.
[6] J. Hopcroft, D. Joseph, S. Whitesides, On the movement of robot arms in 2-dimensional bounded regions, SIAM Journal on Computing 14 (2) (1985) 315–333.
[7] J.E. Hopcroft, D.A. Joseph, S.H. Whitesides, Determining points of a circular region reachable by joints of a robot arm, Technical Report TR82-516, Computer Science Department, Cornell University, 1982.
[8] V. Kantabutra, Motions of a short-linked robot arm in a square, Discrete & Computational Geometry 7 (1992) 69–76.
[9] V. Kantabutra, Reaching a point with an unanchored robot arm in a square, International Journal of Computational Geometry & Applications 7 (6) (1997) 539–549.
[10] V. Kantabutra, S.R. Kosaraju, New algorithms for multilink robot arms, Journal of Computer and System Sciences 32 (1) (1986) 136–153.
[11] W.J. Lenhart, S. Whitesides, Reconfiguring closed polygonal chains in Euclidean *d*-space, Discrete and Computational Geometry 13 (1995) 123–140.
[12] A. Nourollah, M. Razzazi, A New Dynamic Programming Algorithm for Orthogonal Ruler Folding Problem in *d*-Dimensional Space, ICCSA (1), in: Lecture Notes in Computer Science, vol. 4705, Springer, 2007, pp. 15–25. ISBN 978-3-540-74468-9.
[13] A. Nourollah, M. Razzazi, A linear time approximation algorithm for ruler folding problem, Journal of Univarsal Computer Science 14 (4) (2008) 566–574.
[14] A. Nourollah, M. Razzazi, Near optimum folding for a reconfigurable snake-like robot, Advanced Robotics 23 (1–2) (2009) 239–256.
[15] J. O'Rourke, Folding and unfolding in computational geometry, Discrete and Computational Geometry 1763 (Dec.) (1998) 258–266.
[16] Y. Shan, Robot obstacle accomodation: mechanics, in: Control and Applications, Ph.D. Thesis, The University of Michigan, September, 1993.
[17] Y. Shan, Y. Koren, Design and motion planning of a mechanical snake, IEEE Transactions on Systems, Man, and Cybernetics 23 (4) (1993) 1091–1100.
[18] S. Whitesides, Chain Reconfiguration. The Ins and Outs, Ups and Downs of Moving Polygons and Polygonal Linkages, ISAAC, 2001, pp. 1-13.
[19] S. Whitesides, Algorithmic issues in the geometry of planar linkage movement, Australian Computer Journal, Special Issue on Algorithms 24 (2) (1992) 42–50.